COMMENTS ON "A GENERAL FORTRAN EMULATOR FOR IBM 360/370 RANDOM NUMBER GENERATOR 'RANDU'[1]"

BY

DAVID M. FELLOWS

TR75-006, AUGUST 1975

COMMENTS ON

"A GENERAL FORTRAN EMULATOR FOR IBM 360/370 RANDOM NUMBER GENERATOR 'RANDU'[1]"

BY

DAVID M. FELLOWS

School of Computer Science
University of New Brunswick
Fredericton, N.B.

TR75-006, August 1975

Recently Herbst[1] presented a Fortran subroutine, SRANDU, which emulates the subroutine RANDU[2] and that was claimed to be portable among computers with a word length of 32 bits or greater. The routine, as presented by Herbst, is portable in the sense that it will almost certainly compile and execute without exception on any computer with a Fortran compiler. Unfortunately, the sequence of "random" numbers produced will not be the same as that produced by subroutine RANDU executing on an IBM /360 or /370 unless the computer used has a very long word length (e.g. CDC 6400). In addition, when the routine was tested on an IBM /370 using the WATFIV, FORTRAN-G and FORTRAN-H compilers, a compiler related problem manifested itself thus illustrating once again the difficulty of writing portable Fortran programs.

Prior to discussing the problems with SRANDU it is appropriate to specify exactly what the subroutine RANDU produces. RANDU produces two sequences. One is the sequence of integers defined by:

$$n_{i+1} = (n_i * 65539) \bmod 2^{31} \tag{1}$$

given an arbitrary, odd, positive, value of n_0 as a "seed". The values of n_i are perforce odd, positive, and less than 2^{31}. The second sequence is formed by mapping the sequence defined by (1) onto the real interval (0, 1). This is done by approximating

$$x_i = n_i/2^{31} \tag{2}$$

using single precision floating point arithmetic.

Any emulator of RANDU must reproduce the sequence defined by (1) exactly. As Herbst pointed out, the values of x_i as provided by an emulator may vary from those provided by RANDU depending on the word lengths used in the floating point representation of (2). This error will be small and numerically stable and will normally be masked by other round-off errors in the computations in which the x_i are used.

The problems are related to the fact that integer arithmetic is being emulated using floating point arithmetic. It is, of course, quite permissible to use the floating point facilities of a machine and language to perform operations on integers. The results will be exact if the fractional part of the floating point number is sufficiently large to hold all the digits of the integers being used and if care is taken in the source language to see that "small", integer, floating point numbers are represented in such a way that they are represented as small integers in the object code.

In the case of SRANDU, which is reproduced with line numbers added in Figure 1, a problem arises in the use of FLOAT (IX) in L3. FLOAT is a function returning a single precision floating point value. IX is an integer variable which may take on positive odd values in the range 1 to $2^{31}-1$. It requires a minimum of 31 bits for its exact representation. Most computers, CDC 6000 and CYBER 70 series excepted, do not provide 31 bit fractional parts for single precision floating point numbers. Applying the function DBLE to FLOAT (IX) does not alleviate the problem since the digits are lost before conversion to double precision. The obvious way to eliminate the problem would be to use a function DFLOAT (IX) to convert IX to double precision directly.

Unfortunately ANSI Standard Fortran[3] does not require the provision of such a function. To conform to ANSI Standard Fortran the conversion can be done using a simple assignment statement (see Figure 2).

A second problem also occurs in line L3. This is the representation of the integer 65539 as a double precision constant. In writing a truly portable routine this is not a trivial problem. It is necessary that the constant be represented internally exactly (and it can be on machines with suitable word sizes). However, the conversion of REAL and DOUBLE PRECISION constants to internal binary (or hexadecimal or octal) form is subject to round-off error in general. The occurrence and amount of this error is determined by the number to be converted and by the conversion algorithm used. In particular, it was found that the WATFIV compiler introduces round-off error in converting 6.5539D4. Apparently the WATFIV algorithm converts the decimal fraction 6.5539 to a hexadecimal fraction as an intermediate step. The hexadecimal fraction is a truncation of a non-terminating fraction thus introducing round-off error. The FORTRAN-G and FORTRAN-H compilers use a different algorithm which produces an exact conversion in this instance. The ANSI Standard is ambiguous enough that all three compilers comply with it in their DOUBLE PRECISION constant conversion. In the light of the Standard and of typical conversion techniques it appears that the most portable way of expressing the constant is in the form 65539D0. It seems highly unlikely that any compiler purporting to compile ANSI Standard Fortran would introduce round-off error in converting the constant in that form. A similar remark applies to line L4.

SRANDU also contains some redundant statements which obscure its function. Lines L6 and L7 were apparently copied from subroutine RANDU without understanding their purpose. In subroutine RANDU they perform part of the operation of taking the product modulo 2^{31}. In SRANDU this operation is performed by the DMOD function and consequently X and IX will always be positive.

It is perhaps worth noting that the minimum size of the floating point word required by an emulator routine is determined by Y which must hold the product in (1) exactly. Since 65539 requires 17 bits for its exact representation and IX will have 31 bits, their product will produce a maximum of 48 significant bits which will require a minimum of 48 bits in the fractional part of the floating point number.

A subroutine to emulate RANDU, believed to be written in ANSI Standard Fortran, is shown in Figure 2. It overcomes the shortcomings of SRANDU. It should be portable among most computers with 32 bit or larger word sizes, but note the caveats in the comments. Rather than attempt to reproduce the behavior of RANDU exactly, equation (2) was implemented as accurately as possible.

It has been extensively tested on an IBM S/370 computer. The sequence of IY values is identical with those produced by RANDU from the same seed. A sample sequence of 50 floating point numbers was compared with the corresponding sequence produced by RANDU. It has also been tested for short sequences on a Univac 1106 (36 bit word length). On both machines the largest differences found were 1 in the 7th significant digit.

Table 1 gives the first 15 numbers produced by subroutine RANDU using 1 as a seed. It should be used for checking that any RANDU emulator is working properly.

References

1. Herbst, A.F., "A General Fortran Emulator for IBM 360/370 Random Number Generator 'RANDU'", INFOR, 13 (2) 211-212 June 1975.

2. International Business Machines, Systems/360 Scientific Subroutine Package Version III Programmer's Manual, Order No. H20-0205-3, IBM Corp., 1968.

3. American National Standards Institute, USA Standard Fortran X3.9-1966, ANSI, New York, 1966.

TABLE 1. Sequences produced by RANDU.

i	n_i	x_i
0	1	
1	65539	0.3051898E-04
2	393225	0.1831097E-03
3	1769499	0.8239872E-03
4	7077969	0.3295936E-02
5	26542323	0.1235973E-01
6	95552217	0.4449496E-01
7	334432395	0.1557322E 00
8	1146624417	0.5339385E 00
9	1722371299	0.8020415E 00
10	14608041	0.6802399E-02
11	1766175739	0.8224396E 00
12	1875647473	0.8734163E 00
13	1800754131	0.8385414E 00
14	366148473	0.1705011E 00
15	1022489195	0.4761336E 00

FIGURE 1. Herbst's Original Subroutine

```
L1      SUBROUTINE SRANDU (IX,IY,RND)
L2      DOUBLE PRECISION X,Y,DBLE,DMOD
L3      Y = DBLE(FLOAT(IX))*6.5539D + 04
L4      X = DMOD(Y,2.147483648D + 09)
L5      IY = IDINT(X)
L6      IF(IY) 1,2,2
L7    1 IY = IY + 2147483647 + 1
L8    2 TEMP = FLOAT(IY)
L9      RND = TEMP*4.656613E - 10
L10     RETURN
L11     END
```

FIGURE 2. Portable Subroutine to Emulate RANDU.

```
      SUBROUTINE RANDUE(IX,IY,RND)
C
C     ANSI STANDARD FORTRAN EMULATOR FOR
C     RANDU - IBM S/360-370 RANDOM NUMBER
C     GENERATOR.
C     THIS ROUTINE WILL ONLY WORK PROPERLY
C     ON MACHINES/COMPILERS WHICH USE AT
C     LEAST 48 BITS TO STORE THE FRACTIONAL
C     PART OF DOUBLE PRECISION VARIABLES
C     AND AT LEAST 32 BITS TO STORE INTEGER
C     VARIABLES.
C     IF SINGLE PRECISION VARIABLES ARE
C     STORED WITH AT LEAST 48 BITS IN THE
C     FRACTIONAL PART, THEN THE ROUTINE MAY
C     BE MODIFIED TO USE SINGLE PRECISION
C     THROUGHOUT.
C     BEFORE USING THIS ROUTINE PERFORM THE
C     FOLLOWING CHECKS:
C       A) IX=1 RESULTS IN IY=65539
C       B) IX=1146624417 RESULTS IN IY=1722371299
C     IF THE CHECKS DO NOT AGREE YOUR COMPILATION
C     OF THE ROUTINE IS NOT EMULATING RANDU.
C
      DOUBLE PRECISION X,Y,DMOD
      X=IX
      Y=DMOD(X*65539D0,2147483648D0)
      IY=Y
      RND=Y/2147483648D0
      RETURN
      END
```